

(5)

```

# sip client kludge
implement Command;

Mod : con "sipc";
Version : con "SIP/2.0";
Transport := "UDP";

include "sys.m";
sys: Sys;
stderr : ref Sys->FD;

include "draw.m";
include "daytime.m";
include "csget.m";
daytime: Daytime;
include "sh.m";

# Optional audio driver for ephone
# Typically this is in "/prod/shanip/module/UCBAudio.m"
# Use namespace to place it: bind -a /prod/shanip/module /module
include "UCBAudio.m";
ua : UCBAudio;

# Default init values
default_lport : con "5060";
default_rtport : con "3456";
default_rtpport : con "3466";
default_client : con "8089:8089";
default_aproto : con "RTP/AVP";
default_exptime : con "3600";

# RTP and remote RTP ports (default RTCP is 1+)
Rtpport := default_rtport;
Rrtport := default_rtpport;

# Registration expiration
Exptime := default_exptime;

# Proxy/Registrar definition example:
# Proxy : string = "135.1.89.127:5060";
Proxy, Registrar : string;

# Audio protocol selection
Aproto := default_aproto;

# This client local address (or substituted address)
Laddr : string;

active := 0;
Epid := 0;

# To reset the client process
Args : list of string;

# To delay first registration
Zreg := 0;

# Debug level
Dbg := 0;
# Verbose level -- message sent/received contents only
Vbs := 0;

init(ctxt : ref Draw->Context, args : list of string)
{
    Args = args;
    sys = load Sys Sys->PATH;
    stderr = sys->filides(2);
    daytime = load Daytime Daytime->PATH;
    if(daytime == nil) {
        sys->fprintf(stderr, Mod+": load %s: %r\n", Daytime->PATH);
        return;
    }
    ua = load UCBAudio UCBAudio->PATH;
    if (ua == nil) {
        (ok, reason) := ua->initialize();
        if (ok < 0) sys->fprintf(stderr, Mod+": %s\n", reason);
    }

    cs := load CsGet CsGet->PATH;
    (nil, Laddr, nil) = cs->hostinfo(nil);
    if (Laddr == nil) return;
    if (Dbg) sys->print(Mod+": local address: %s\n", Laddr);

    if (args != nil)
        args = tl args;

    ok : int;
    client : string;
    (ok, client, args) = parseopt(args);
    if (!ok) return;

    if (client == nil) client = default_client;
    if (numberp(client)) client += "@:" + client;
    client = thisclient(client);
    sys->print(Mod+": this client: %s\n", client);

    C = ref Calls(nil, nil, nil);

    if (args != nil)
        clients = args;
    else if (Registrar == nil) {
        sys->print(Mod+": using Styx locator for SIP clients\n");
        readclients();
        registerclient(client);
    }

    if (!siplisten(client)) return;
    if (Registrar != nil && !Zreg)
        C.this = register(client, nil);

    ch := chan of int;
    spawn sound(ch);
}

```

```

Spid = <- ch;
if (Dbg) sys->print(Mod+": sound process %d\n", Spid);
spawn rcmd(ctxt, client_nonet(client), ch);
pid := <- ch;
if (Dbg) sys->print(Mod+": command process %d\n", pid);
if (ua != nil) {
    spawn listenkeys(ch);
    Epid = <- ch;
    if (Dbg) sys->print(Mod+": ephone process %d\n", Epid);
}
}

usage()
{
    sys->print("usage: sip<\t[options] [this_client] [other_client]... [more client entries]\n\t\t\t-a\t-- specify audio protocol: ud\t"
}

parseopt(args : list of string) : (int, string, list of string)
{
    client : string;
    atcp := 0;
out:
    for (; args != nil; args = tl args) {
        opt := hd args;
        case opt {
            "??" or "-?" or "help" or "--help" =>      usage(); return (0, nil, args);
            "-a" => {
                args = tl args;
                if (args != nil) {
                    case hd args {
                        "tcp" or "TCP" or "RTP/TCP" or "RTP/TCP/AVP" => Aproto = "RTP/TCP"; atcp = 1;
                        * => Aproto = default_aproto;
                    }
                    if (Dbg) sys->print(Mod+": audio protocol set to %s\n", Aproto);
                }
            }
            "-b" => {
                args = tl args;
                if (args != nil) {
                    (rt, rr) := expand2t(hd args, "/");
                    if (rt != nil) {
                        Rrport = rt;
                        if (rr != nil) Rrport = rr;
                        else Rrport = string (int rt + 10);
                    }
                    else {
                        Rtpport = default_rtpport;
                        Rrport = default_rrport;
                    }
                }
                if (Dbg) sys->print(Mod+": audio RTP ports set to %s/%s\n", Rtpport, Rrport);
            }
            "-d" => Dbg++;
            "-v" => Vbs++;
            "-l" => {
                args = tl args;
                if (args != nil) {
                    val := hd args;
                    sys->print(Mod+": local address set %s -> %s\n", Laddr, val);
                    Laddr = val;
                }
            }
            "-p" => {
                args = tl args;
                if (args != nil) {
                    val := hd args;
                    if (val == "-") {
                        proxies := readlist("./services/server/sip_proxies");
                        if (proxies != nil) Proxy = Registrar = hd proxies;
                        else sys->fprint(stderr, Mod+": empty ./services/server/sip_proxies");
                    }
                    else {
                        Proxy = val;
                        if (Registrar == nil) Registrar = val;
                    }
                    if (Dbg) sys->print(Mod+": proxy set to %s\n", Proxy);
                }
            }
            "-r" => {
                args = tl args;
                if (args != nil) {
                    Registrar = hd args;
                    if (Dbg) sys->print(Mod+": registrar set to %s\n", Registrar);
                }
            }
            "-z" => {
                Zreg++;
                if (Dbg) sys->print(Mod+": registration postponed\n");
            }
            "-o" or "-ol" => {
                args = tl args;
                if (args != nil) {
                    Port_offset = int hd args;
                    if (Dbg) sys->print(Mod+": port offset set to %d\n", Port_offset);
                }
            }
            "-ao" or "-o2" => {
                args = tl args;
                if (args != nil) {
                    Aport_offset = int hd args;
                    if (Dbg) sys->print(Mod+": announced port offset set to %d\n", Aport_offset);
                }
            }
            * => {
                if (opt != nil) {
                    if (opt[0] == 'S') {
                        nc := int opt[1];
                        client = nth(nc, readlist("./services/config/sip_phones"));
                    }
                    else if (client == nil) client = opt;
                    if (Dbg) sys->print(Mod+": client set to %s\n", client);
                    args = tl args;
                }
                break out;
            }
        }
    }
    if (atcp) tcpaudio();
    return (1, client, args);
}

```

```

sip_Aport := default_lport;
siplisten(client : string) : int
{
    ch := chan of int;
    ok : int; conn : Sys->Connection;
    if (client != nil) {
        (nil, nil, port, nil) := expandnet(client);
        net := downcase(Transport);
        sip_Aport = port;
        (ok, conn) = announce(net, "", port);
        if (ok < 0) return 0;
        spawn listen(client, conn, ch);
        active = <- ch;
        if (Dbg) sys->print(Mod+": listen process %d\n", active);
        if (!Monpid) {
            spawn monitor(client, ch);
            Monpid = <- ch;
            if (Dbg) sys->print(Mod+": monitor process %d\n", Monpid);
        }
    }
    return 1;
}

Monpid := 0;
monitor(client : string, ch : chan of int)
{
    ch <- sys->pctl(0, nil);
    while(Monpid) {
        sys->sleep(Timeout);
        if (!Monpid) return;
        for (l := C.clist; l != nil; l = tl l) {
            c := hd l;
            if (c == nil || !c.expire) continue;
            if (c.expire <= time()) {
                c.expire = 0;
                c.resend(client);
            }
        }
    }
}

Lastrt := 0;
restartsip(client : string, c : ref Call) : int
{
    if ((nt := time()) > Lastrt) {
        Lastrt = nt + Timeout;
        sys->fprint(stderr, Mod+": would restart sip listener\n");
        nt = 0;
        return 1;
    }

    pid := 0;
    if (!nt) {
        pid = active;
        active = 0;
    }
    if (c.conn != nil) c.conn.dfd = nil;
    c.conn = nil;
    if (!nt) {
        sys->sleep(100);
        kill(pid);
    }
    if (C.this != nil) C.this.conn = nil;
    if (C.recv != nil) C.recv.conn = nil;
    if (!nt) {
        sys->sleep(100);
        if(siplisten(client)) return 1;
        else sys->fprint(stderr, Mod+": cannot restart sip listener\n");
    }
    return 0;
}

include "kill.m";
kp : Kill;

kill(pid : int)
{
    if (kp == nil) kp = load Kill Kill->PATH;
    kp->killpid(string pid, array of byte "kill");
}

cleanup()
{
    killsound();
    pid := Monpid;
    Monpid = 0;
    sys->sleep(100);
    kill(pid);
    for (l := C.clist; l != nil; l = tl l) {
        c := hd l;
        if (c != nil && c.session != nil)
            c.session.endaudio();
    }
    if (pid = active) {
        active = 0;
        sys->sleep(100);
        kill(pid);
    }
    if (pid = Epid) {
        Epid = 0;
        sys->sleep(100);
        kill(pid);
    }
    cleanClist(l);
    Dbg = 0;
}

# /tmp/sipcmd channel to control client from another program
# this does not deal with digit collection yet...

@sipsrv : con "sipcmd";
sipsrv : con "sc";
mp : con "/tmp";

rcmd(ctxt : ref Draw->Context, client : string, rch : chan of int)
{
    sys->bind("#$", mp, sys->MBEFORE);
    ch := sys->file2chan(mp, sipsrv);
    if (ch == nil) {

```

```

rch <- = 0;
sys->fprintf(stderr, Mod+": file2chan %s/%s %r\n", mp, sipsrv);
return;
}
else rch <- = sys->pctl(0, nil);

if (Dbg) sys->print(Mod+": %s/%s is the command interpreter\n", mp, sipsrv);

run := 1;
reset := 0;
while (run) {
    alt {
        (o, data, fid, wc) := <- ch.write =>
            if (data != nil && wc != nil) {
                sdata := string data;
                if (Dbg) sys->print(Mod:> "%s", sdata);
                if (sdata == "restart") {
                    run = 0;
                    restartdevice(ctxt);
                }
                else if (sdata == "reset") reset = !(run = 0);
                else run = sipdo(client, sdata);
                wc <- = (len data, nil);
            }
        (o, n, fid, rc) := <- ch.read =>
            data := array of byte `sip commands - write help to read the menu`;
            if (rc != nil && n > 0) {
                if (n < len data) data = data[0:n];
                rc <- = (data, "");
            }
            else if (rc != nil) rc <- = (nil, "");
    }
}

sys->umount(*#s, mp);
cleanup();
if (reset) spawn init(ctxt, Args);
}

Call : adt
{
    conn : ref Sys->Connection;
    path : ref Path;
    frum : string;
    tu : string;
    callid : string;
    cseq : string;
    state : string;
    session : ref Session;
    expire : int;
    msg : string;
    initied : int;           # True when call initiated on this end
    store : fn(c : self ref Call, c2 : ref Call);
    send : fn(c : self ref Call, client : string) : ref Call;
    resend : fn(c : self ref Call, client : string);
    resendmsg : fn(c : self ref Call, client, msg : string);
    nextstate : fn(c : self ref Call, client : string);
    disconnect : fn(c : self ref Call, client, end : string) : ref Call;
    addsession : fn(c : self ref Call, sid, data : string) : int;
    addedsession : fn(c : self ref Call) : int;
    stateinfo : fn(c : self ref Call) : (string, int, string);
    activep : fn(c : self ref Call) : int;
    endp : fn(c : self ref Call) : int;
    registerp : fn(c : self ref Call) : int;
};

Call.stateinfo(c : self ref Call) : (string, int, string)
{
    if (c == nil) return (nil, 0, nil);
    s := c.state;
    token, num, msg : string;
    (nil, ls) := sys->tokenize(s, " \t");
    if (ls != nil) {
        if (tl ls != nil) {
            token = hd ls;
            num = hd tl ls;
            for(l := tl tl ls; l != nil; l = tl l) {
                msg += hd l;
                if (tl l != nil) msg += " ";
            }
        }
        else token = hd ls;
    }
    else token = c.state;
    n := 0;
    if (num != nil)
        if (numberp(num)) n = int num;
    else sys->fprintf(stderr, Mod+": unexpected state %s %s\n", token, num);
    if (Dbg > 2) sys->print(Mod+": stateinfo -> (%s, %d, %s)\n", token, n, msg);
    return (token, n, msg);
}

Call.store(c1 : self ref Call, c2 : ref Call)
{
    c1.state = c2.state;
    # preserve recorded route for future requests (e.g. ack)
    if (c2.path != nil) {
        if (c1.path == nil || c2.path.record != nil) {
            if (Dbg) sys->print(Mod+": storing new path in call (record route) %s\n", c1.callid);
        # assume contact not changed
            c1.path = c2.path;
        } else if (c2.path.route != nil) {
            if (Dbg) sys->print(Mod+": storing new path in call (route) %s\n", c1.callid);
            c1.path = c2.path;
            route := c2.path.route;
            cont := c2.path.contact;
            if (cont != nil) {
                cont = mksipurl(sipurival(cont));
                if (Dbg) sys->print(Mod+": rewriting route in call %s using %s\n", c1.callid, cont);
                r := cont :: nil;
                for (; route != nil; route = tl route)
                    if (tl route != nil) r = hd route :: r;
                c1.path.route = r;
            }
            via := c2.path.via;
            if (via != nil && cont != nil) {
                c1.path.contact = cont;
                if (Dbg) sys->print(Mod+": rewriting via in call %s using %s\n", c1.callid, cont);
                v := mkvia(cont) :: nil;
                for (; via != nil; via = tl via)
                    if (tl via != nil) v = hd via :: v;
                c1.path.via = v;
            }
        }
    }
}

```

```

        }

    }

    c1.frum = c2.frum;
    c1.tu = c2.tu;
    c1.initied = c2.initied;
    c1.cseq = c2.cseq;
}

Session : adt
{
    sid : string;
    data : string;
    rdata : list of string;
    audio : ref Audio;
    endaudio : fn(s : self ref Session);
    startaudio : fn(s : self ref Session, tipe : int);
    dialaudio : fn(s : self ref Session);
    announceaudio : fn(s : self ref Session);
};

Audio : adt
{
    addr1 : string;
    addr2 : string;
    tipe : int;
    conn1 : ref Sys->Connection;
    conn2 : ref Sys->Connection;
    listen : int;
    spead : int;
    rtcp1 : int;
    rtcp2 : int;
    cconn1 : ref Sys->Connection;
    cconn2 : ref Sys->Connection;
    size : int;
    busy : int;
};

Calls : adt
{
    clist : list of ref Call;
    this : ref Call;
    recv : ref Call;
    find : fn(c1 : self ref Calls, id : string) : ref Call;
    item : fn(c1 : self ref Calls, n : int) : ref Call;
    next : fn(c1 : self ref Calls) : ref Call;
    take : fn(c1 : self ref Calls, c : ref Call);
    add : fn(c1 : self ref Calls, c : ref Call) : int;
    rem : fn(c1 : self ref Calls, c : ref Call) : int;
    remrecv : fn(c1 : self ref Calls, c : ref Call) : int;
    print : fn(c1 : self ref Calls);
};

# Master call list
C : ref Calls;

Calls.print(c1 : self ref Calls)
{
    sys->print(Mod+": call list:\n");
    i := 0;
    ct := C.this;
    cr := C.recv;
    for (l := c1.clist; l != nil; l = tl l) {
        c := hd l;
        mode : string;
        if (ct == c) {mode = "this call"; ct = nil;}
        else if (C.recv == c) {mode = "recv call"; cr = nil;}
        else mode = "";
        sys->print("%d: call %s %s %s\n", ++i, c.callid, c.state, mode);
    }
    if (ct != nil) sys->print("%: idle %s %s this call\n", ct.callid, ct.state);
    if (cr != nil) sys->print("%: idle %s %s recv call\n", cr.callid, cr.state);
}

Calls.find(c1 : self ref Calls, id : string) : ref Call
{
    for (l := c1.clist; l != nil; l = tl l)
        if ((hd l).callid == id) return hd l;
    return nil;
}

Calls.item(c1 : self ref Calls, n : int) : ref Call
{
    i := 1;
    for (l := c1.clist; l != nil; l = tl l)
        if (i == n) return hd l;
        else i++;
    return nil;
}

Calls.next(c1 : self ref Calls) : ref Call
{
    c := c1.this;
    for (l := c1.clist; l != nil; l = tl l)
        if (hd l == c)
            if (tl l != nil) return hd tl l;
            else return hd c1.clist;
    return nil;
}

Calls.take(c1 : self ref Calls, c : ref Call)
{
    if (c != nil) {
        if (c1.clist != nil && c1.this != nil && c1.this.callid != c.callid) {
            if (Proxy != nil && c.conn == nil && c1.this.conn != nil) c.conn = c1.this.conn;
            if (Dbg > 1) sys->print(Mod+": switching call %s -> %s\n", c1.this.callid, c.callid);
        }
        pc := c1.find(c.callid);
        if (pc != nil) {
            if (pc == c) {
                c1.this = c;
                c1.remrecv(c);
                return;
            }
            else c1.rem(pc);
        }
        c1.clist = (c1.this = c) :: c1.clist;
    }
}

```

```

Calls.add(cl : self ref Calls, c : ref Call)
{
    if (c != nil) {
        pc := cl.find(c.callid);
        if (pc != nil) cl.rem(pc);
        cl.clist = c :: cl.clist;
    }
}

Calls.remrecv(cl : self ref Calls, c : ref Call) : int
{
    if (cl.recv != nil && cl.recv.callid == c.callid) {
        cl.recv = nil;
        return 1;
    }
    return 0;
}

Calls.rem(cl : self ref Calls, c : ref Call) : int
{
    if (c == nil) return 0;
    if (Dbg > 1) sys->print(Mod+": removing call %s\n", c.callid);
    n := 0;
    r : list of ref Call;
    for (l := cl.clist; l != nil; l = tl l)
        if (hd l != c && (hd l).callid != c.callid) r = hd l :: r;
        else n++;
    cl.clist = reversec(r);
    if (cl.this == c)
        if (cl.clist != nil) cl.this = hd cl.clist;
        else cl.this = nil;
    if (cl.recv == c) cl.recv = nil;
    cl.remrecv(c);
    return n;
}

reversec(l : list of ref Call) : list of ref Call
{
    r : list of ref Call;
    for(; l != nil; l = tl l) r = hd l :: r;
    return r;
}

sipdo(client, cmd : string) : int
{
    c := C.this;
    (nil, cl) := sys->tokenize(cmd, " \t\r\n");
    if (cl == nil) return 1;
    Sch <- ("", 0);
    case hd cl {
        "a" => {
            if (tl cl != nil) {
                line := hd tl cl;
                called : string;
                if ("Proxy" == nil) {
                    if (Ldomain != nil && pos('@', line) < 0)
                        called = findclient(line+Ldomain);
                    if (called == nil)
                        called = findclient(line);
                }
                else if (Ldomain != nil && pos('@', line) < 0) called = line + Ldomain;
                else called = line;

                if (called == client) {
                    sys->fprintf(stderr, Mod+": calling self %s\n", client);
                    Sch <- ("x", -1);
                }
                else if (called != nil) {
                    if (Dbg) sys->print(Mod+": calling %s\n", called);
                    c = connect(client, called, c);
                    C.take(c);
                    Sch <- ("w", -1);
                }
                else {
                    sys->fprintf(stderr, Mod+": client not found at line %s\n", line);
                    Sch <- ("x", -1);
                }
            }
            else if (c != nil && !c.registerp()) {
                if (c.state == "INVITE 180 Ringing") {
                    c.state = "INVITE 200 OK";
                    c.send(client);
                } else if (start("INVITE ", c.state))
                    c.nextstate(client);
                else
                    if (Dbg) sys->print(Mod+": in call %s %s\n", c.callid, c.state);
            }
            else {
                sys->fprintf(stderr, Mod+": missing line number\n");
                Sch <- ("x", -1);
            }
            return 1;
        }
        "f" => {
            if (tl cl != nil) {
                cn := int hd tl cl;
                c = C.item(cn);
                if (c != nil) C.take(c);
                else sys->fprintf(stderr, Mod+": call number %d not found\n", cn);
            }
            else {
                c = C.next();
                if (c != nil) C.take(c);
            }
            return 1;
        }
        "1" => {
            C.print();
            return 1;
        }
        "r" => {
            c = register(client, c);
            if (C.this == nil) C.this = c;
            return 1;
        }
        "z" => {
            if (c == nil) c = C.this = C.recv;
            if (c == nil) sys->fprintf(stderr, Mod+": no current call\n");
            else {
                if (c.state == "INVITE 200 OK" || c.state == "ACK") {

```

```

        c.state = "ACK";
        c = c.disconnect(client, "BYE");
        C.take(c);
        C.rem(c);
        return 1;
    }
    else {
        if (!start("BYE", c.state) && !start("REGISTER", c.state)) {
            c = c.disconnect(client, "CANCEL");
            C.take(c);
        }
        C.rem(c);
        return 1;
    }
}
S.s = S.d = nil;
if (c != nil) {
    c.inited = 0;
    C.rem(c);
}
cleanList();
}
*q => return 0;
* => {
    cmd := hd cl;
    if (cmd != nil) {
        case cmd[0] {
            '-' or '=' => {
                eq1 := cmd[0] == '=';
                cl = tl cl;
                if ((cmd = cmd[1:]) != nil) cl = cmd :: cl;
                if (eq1) cl = "=" :: cl;
                test(cl);
                return 1;
            }
        }
    }
    sys->fprintf(stderr, Mod+": a <number>, f, l, q, r, z, and (-, =)[cmd] : are supported commands\n");
}
return 1;
}

include "qidcmp.m";
qc : Qidcmp;
Cdir : import qc;

clients : list of string;
Spath : con "/services/server/sip_clients";
Sqid : ref Cdir;

# load the last record of all clients that connected via sip
readclients() : list of string
{
    if (Sqid == nil) {
        qc = load Qidcmp Qidcmp->PATH;
        if (qc == nil) {
            sys->fprintf(stderr, Mod+": %s %r\n", Qidcmp->PATH);
            return nil;
        }
        qc->init(nil, nil);
        Sqid = ref Cdir(nil, Qidcmp->SAME);
    }
    if (Sqid.fcmp(Spath)) {
        if (Dbg) sys->print(Mod+": updating Styx SIP clients\n");
        clients = readlist(Spath);
    }
    return clients;
}

addclients(client : string)
{
    if (Dbg) sys->print(Mod+": adding SIP client locator %s\n", client);
    fappend(Spath, client);
    readclients();
}

replaceclients(new, old : string)
{
    if (Dbg) sys->print(Mod+": updating SIP client locator %s -> %s\n", old, new);
    r : list of string;
    for (l := clients; l != nil; l = tl l) {
        if (hd l == old) r = new :: r;
        else r = hd l :: r;
    }
    clients = reverse(r);
    writelist(Spath, clients);
    readclients();
}

registerclient(client : string)
{
    (l, a, p) := expand(client);
    host := findclient(l);
    if (host == nil)
        addclients(client);
    else if (host != client)
        replaceclients(client, host);
}

findclient(line : string) : string
{
    readclients();
    for(l := clients; l != nil; l = tl l) {
        (num, nil, nil) := expand(hd l);
        if (num == line) return hd l;
    }
    return nil;
}

# Local domain starts with @
Ldomain : string;

thisclient(client : string) : string
{
    (who, laddr, port) := expand(client);
    if ((p := pos('@', who)) >= 0) {
        Ldomain = who[p+1];
        if (Dbg) sys->print(Mod+": local domain %s\n", Ldomain);
    }
}

```

```

        if (Laddr != laddr && addressp(laddr) == 1) {
            sys->print(Mod+": local address substitute: %s -> %s\n", Laddr, laddr);
            Laddr = laddr;
        }
        return who+"@"+laddr+":"+thisport(port);
    }

thisport(p : string) : string
{
    (n, lp) := sys->tokenize(p, "/");
    case n {
        1 => Transport = "UDP";
        2 => (Transport = upcase(fd lp); p = fd tl lp;)
        * => sys->fprint(stderr, Mod+": unexpected port argument %s\n", p);
    }
    return downcase(Transport)+"/"+p;
}

ntime() : int
{
    return int 1e+09 + daytime->now();
}

rtime() : int
{
    return daytime->now();
}

Timeout := 16000;
etime() : int
{
    return sys->millisec() + Timeout;
}

time() : int
{
    return sys->millisec();
}

explicitport(entry, port : string) : string
{
    if (port != default_lport) entry += ":"+port;
    else if (Proxy != nil) {
        (nil, nil, pp, nil) := expandnet(Proxy);
        if (port != pp) entry += ":"+port;
    }
    if (Dbg) sys->print(Mod+": explicitport() -> %s\n", entry);
    return entry;
}

proxy(client : string) : string
{
    if (Proxy == nil) return client;
    return Proxy;
}

proxytype() : string
{
    if (Proxy != nil) {
        (t, nil, nil) := expand(Proxy);
        return t;
    }
    return nil;
}

mkvia(client : string) : string
{
    if (client != nil) {
        (nil, vaddr, vport, net) := expandnet(client);
        # remove the line@ since it crashes vovida phones
        #return "+Version"+"/"+upcase(net)+" "+client;
        entry := "+Version"+"/"+upcase(net)+" "+vaddr;
        return explicitport(entry, vport);
    }
    return nil;
}

viaproxy(proxy, contact : string, via : list of string) : list of string
{
    if (proxy != nil) return mkvia(proxy) :: via;
    else if (contact != nil) return mkvia(contact) :: via;
    else return via;
}

numberp(s : string) : int
{
    for (i := 0; i < len s; i++) {
        c := s[i];
        if (c < '0' || c > '9') return 0;
    }
    return 1;
}

# may be an ip address field
addrfp(s : string) : int
{
    for (i := 0; i < len s; i++) {
        c := s[i];
        if ((c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || c == '-') continue;
        else return 0;
    }
    return 1;
}

# return 1 if numeric address > 1 if hostname 0 if neither
addressp(s : string) : int
{
    (n, l) := sys->tokenize(s, ".");
    r := l;
    if (n <= 1) {
        if (l != nil && addrfp(fd l)) ++r;
    }
    else for (; l != nil; l = tl l)
        if (!numberp(fd l))
            if (addrfp(fd l)) ++r;
        else return 0;
    return r;
}

```

```

expand(client : string) : (string, string, string)
{
    (n, la) := sys->tokenize(client, "*");
    pa := client;
    line, addr, port : string;
    if (n >= 2) {
        if (n > 2)
            for(l := 1; l != nil; l = tl l) {
                line += hd l;
                if (tl l != nil)
                    if (tl tl l != nil) line += "*";
                    else { pa = hd tl l; break; }
            }
        else {
            line = hd la;
            pa = hd tl la;
        }
    }
    else if (n) (line = "()", pa = hd la);

    (n, la) = sys->tokenize(pa, ":");
    if (n >= 2) {
        if (addressp(hd la)) addr = hd la;
        else {
            addr = Laddr;
            if (line == nil) line = hd la;
        }
        port = hd tl la;
    }
    else {
        if (line == nil) {
            line = pa;
            addr = Laddr;
        }
        else {
            addr = pa;
            if (numberp(addr)) addr = Laddr;
        }
        port = default_lport;
    }
    if (line == "()") line = nil;
    if (Dbg > 2) sys->print(Mod+": expand(%s -> (%s, %s, %s)\n", client, line, addr, port);
    return (line, addr, port);
}

Call.registerp(c : self ref Call) : int
{
    if (c == nil) return 0;
    return start("REGISTER", c.state);
}

register(frum : string, c : ref Call) : ref Call
{
    if (Registrar == nil)
        Registrar = Proxy;
    if (Registrar == nil)
        sys->fprint(stderr, Mod+": no registrar nor proxy defined\n");
    else {
        reg := Registrar;
        rconn : ref Sys->Connection;
        if (c != nil && (c.registerp() || (Proxy != nil && reg == Proxy)))
            rconn = c.conn;
        if (rconn == nil) {
            (vline, vaddr, vport, net) := expandnet(reg);
            if (Dbg) sys->print(Mod+": connect to %s at %s!%s!%s\n", vport, net, vaddr, vport);
            (ok, conn) := dial(net, vaddr, vport, localport(frum, vport, vaddr));
            rconn = ref conn;
            if (ok < 0) return nil;
        }
        callid := sid2callid(string ntime())+"@"+Laddr;
        path := ref Path(nil, viaproxy(Proxy, nil, mkvia(frum) :: nil), nil, nil);
        frum = client_nonet(frum);
        if (c != nil) {
            c = ref Call(rconn, path, frum, reg, callid, nil, "REGISTER", nil, 0, nil, 0);
            c.send(frum);
            c.expire = int Exptime;
            return c.send(frum);
        }
        else {
            c = ref Call(rconn, path, frum, reg, callid, nil, "REGISTER", nil, int Exptime, nil, 0);
            return c.send(frum);
        }
    }
    return c;
}

connect(frum, tu : string, c : ref Call) : ref Call
{
    rconn : ref Sys->Connection;
    if (c.registerp() && Proxy != nil && Proxy == Registrar)
        rconn = c.conn;
    if (rconn == nil) {
        (vline, vaddr, vport, net) := expandnet(proxy(tu));
        if (Dbg) sys->print(Mod+": connect to %s at %s!%s!%s\n", vport, net, vaddr, vport);
        (ok, conn) := dial(net, vaddr, vport, localport(frum, vport, vaddr));
        if (ok < 0) return nil;
        rconn = ref conn;
    }
    callid := sid2callid(string ntime())+"@"+Laddr;
    path := ref Path(nil, viaproxy(Proxy, tu, mkvia(frum) :: nil), nil, nil);
    frum = client_nonet(frum);
    tu = client_nonet(tu);
    c = ref Call(rconn, path, frum, tu, callid, nil, "INVITE", nil, etime(), nil, 1);
    return c.send(frum);
}

client_nonet(s : string) : string
{
    (n, a, p, nil) := expandnet(s);
    return n+"@"+a+":"+p;
}

# support 4567@1.2.3.4:tcp/5566 format
expandnet(s : string) : (string, string, string, string)
{
    (n, a, p) := expand(s);
    (net, port) := netport(p);
    return (n, a, port, net);
}

```

```

# support tcp/5060 format in proxy and client definitions
netport(np : string) : (string, string)
{
    (net, p) := expand2t(np, "/");
    if (p != nil) np = p;
    else net = downcase(Transport);
    return (net, np);
}

# first port offset -- avoid port numbering collision on same host
Port_offset := 0;

# second port offset -- if port == the announced port
Aport_offset := 0;
#Aport_offset := 30000;

localport(client, port, taddr : string) : string
{
    if (Port_offset) port = string (int port + Port_offset);
    (nil, nil, cpport) := expand(client);
    # we announced or already used this port -- change it
    if (cpport == port || taddr == Laddr) return "1"+port;
    else if (port == sip_Aport && Aport_offset) return string (Aport_offset + int port);
    else return port;
}

Call.disconnect(c : self ref Call, client : string, end : string) : ref Call
{
    c.state = end;
    if(c.session != nil) c.session.endaudio();
    c = c.send(client);
    c.initied = 0;
    # This should not be need - possible bug in udp stack...
    #restartsip(client, c);
    return c;
}

Sipmethods : list of string;
sipmethodp(s : string) : int
{
    if (Sipmethods == nil) Sipmethods = "REGISTER" :: "OPTIONS" :: "INVITE" :: "ACK" :: "BYE" :: "CANCEL" :: nil;
    for(l : Sipmethods; l != nil; l = tl l)
        if (start(hd l, s)) return l;
    return 0;
}

Call.endp(c : self ref Call) : int
{
    s := c.state;
    return s == "CANCEL" || s == "BYE" || start("BYE ", s);
}

# Type of message sent
# 0 long form
# 1 short form
Small := 0;
field(f : string) : string
{
    if (Small)
        case f {
            "From" => f = "f";
            "To" => f = "t";
            "Call-ID" => f = "i";
            "Via" => f = "v";
            "Content-Encoding" => f = "e";
            "Content-Length" => f = "l";
            "Content-Type" => f = "c";
            "Contact" => f = "m";
            "Subject" => f = "s";
            # discrepancies from vovida -> from LSS and 3com
            "Cseq" => f = "CSeq";
        }
    return f;
}

Call.send(c : self ref Call, client : string) : ref Call
{
    (method, code, reason) := c.stateinfo();
    if (!sipmethodp(method)) {
        sys->print(stderr, Mod+": unknown SIP event %s\n", method);
        return nil;
    }
    if (Dbg) sys->print(Mod+": current state %d %s\n", method, code, reason);
    frum := c.frum;
    tu := c.tu;
    callid := c.callid;
    if (c.callid == nil) sys->print(stderr, Mod+": missing callid in call to %s\n", tu);

    (lline, laddr, lport, nil) := expandnet(client);
    (fline, faddr, fport) := expand(frum);
    (tline, taddr, tport) := expand(tu);

    # contact
    cont : string;
    if (method == "ACK" && c.path != nil && c.path.contact != nil)
        cont = sipurival(c.path.contact);
    else cont = explicitport(laddr, lport);

    orig, dest : string;
    if (pos('0', fline) >= 0) {
        orig = fline;
        (fline, nil) = expand2t(fline, "0");
    }
    else if (fline != nil)
        orig = explicitport(fline+"0"+faddr, fport);
    else orig = explicitport(faddr, fport);

    if (pos('0', tline) >= 0) {
        dest = tline;
        (tline, nil) = expand2t(tline, "0");
    }
    else if (tline != nil)
        dest = explicitport(tline+"0"+taddr, tport);
    else dest = explicitport(taddr, tport);

    header, data : string;
    # This was to talk to vovida
    addp := !c.initied;
}

```

```

addp = 0;
if (!code) {
    if (c.registerp()) addp = 0;
    # rfc2543 line 1754
    if (method == "REGISTER" && Ldomain != nil) header += method+" sip:"+Ldomain[1];
    else header += method+" sip:"+add_lport(dest, addp);
    if (addp) header += ";user=phone";
    header += " ";
}

header += Version;
if (code) header += sys->sprint(" %d %s", code, reason);
header += "\r\n";
record := c.path.record;
route := c.path.route;
if (record != nil) {
    header += field("Record-Route")+": ";
    for (r := record; r != nil; r = tl r) {
        header += mksipurl(hd r);
        if (tl r != nil) header += ", ";
    }
    header += "\r\n";
    if (route == nil) sys->fprint(stderr, Mod+": missing route field with record-route\r\n");
}

if (route != nil) {
    for (r := route; r != nil; r = tl r)
        header += field("Route")+": "+mksipurl(hd r)+"\r\n";
}

##compact form
header += field("Route")+": ";
for (r := route; r != nil; r = tl r) {
    header += mksipurl(hd r);
    if (tl r != nil) header += ", ";
}
header += "\r\n";

via := c.path.via;
# this is a response (route is on)
if (method == "ACK") {
    if (Proxy != nil && len via < 2) {
        sys->fprint(stderr, Mod+": response missing via field (%s)\n", method);
        via = viaproxy(Proxy, nil, mkvia(cont) :: nil);
    }
}

if (via != nil && tl via != nil) {
    if (!code && method != "ACK") via = tl via;
    for (; via != nil; via = tl via)
        header += field("Via")+": "+hd via+"\r\n";
}

else header += field("Via")+": "+mkvia(cont)+"\r\n";

# disable this now - was needed to talk to Vovida 1.7
addp = 0;
sipo, sipd : string;
if (code == 200 && method == "BYE") {
    sipo = "<sip:" + orig + ">";
    sipd = "<sip:" + add_lport(dest, addp) + ";user=phone>";
    sipd = "<sip:" + add_lport(dest, addp) + ">";
}
else if (c.registerp()) sipo = sipd = "<sip:" + orig + ">";
else {
    sipo = fline+_phone<sip:" + add_lport(orig, addp) + ">";
    sipd = fline+<sip:" + add_lport(dest, addp) + ";user=phone>";
    sipo = fline+<sip:" + add_lport(orig, addp) + ">";
    sipd = fline+<sip:" + add_lport(dest, addp) + ">";
    sipo = "<sip:" + add_lport(orig, addp) + ">";
    sipd = "<sip:" + add_lport(dest, addp) + ">";
}

header += field("From")+": "+sipo+"\r\n";
header += field("To")+": "+sipd+"\r\n";

# fix first time call id (now that we preserve @)
if (pos('@', callid) < 0) callid += "@"+faddr;
header += field("Call-ID")+": "+ callid +"\r\n";
restart := 0;
cseq := c.cseq;
if (cseq == nil || method == "BYE") {
    seqn := 1;
    if (method == "BYE") {
        seqn++;
        if (code == 200) restart = c.inited;
    }
    cseq = string seqn+" "+method;
    c.cseq = cseq;
}
header += field("CSeq")+": "+cseq+"\r\n";

curl : string;
if (proxypipe) == "lss" curl = "<sip:" + cont + ">";
else {
    if (pos('@', lline) >= 0) (lline, nil) = expand2t(lline, "@");
    curl = "<sip:" + lline + "@" + cont + ">";
}

if (method != "REGISTER") header += field("Contact")+": "+curl+"\r\n";
if (!code && method == "INVITE") {
    header += "User-Agent: Inferno Webphone 2630\r\n";
    header += field("Subject")+": Inferno Webphone INVITE\r\n";
    header += field("Content-Type")+": application/sdp\r\n";
}
if (code == 200 && method == "INVITE") {
    header += field("Content-Type")+": application/sdp\r\n";
}
if (method == "REGISTER") {
    if (!c.expire) header += field("Contact")+": "\r\nExpires: 0\r\n";
    else {
        header += field("Contact")+": "+curl;
        if (Transport != "UDP") header += ";" + transport = downcase(Transport);
        header += "\r\nExpires: "+string c.expire+"\r\n";
    }
    c.expire = 0;
}
header += field("Content-Length")+": ";

csp := 0;
if (!code || code == 200 && method == "INVITE") {
    rtpport := Rtpport;
    daddr := laddr; # was faddr
    if (code == 200) {
        rtpport = Rtpport;
    }
}

```

```

        daddr = derive_taddr(c);
    }
    sid : string;
    if (c.session != nil) sid = c.session.sid;
    else sid = callid2sid(callid);
    data += "v=0\r\no=- "+sid+" IN IP4 "+daddr+"\r\n";
    data += "v=0\r\no=username "+sid+" "+sid+" IN IP4 "+daddr+"\r\n";
    data += "v=0\r\no=username 0 0 IN IP4 "+daddr+"\r\n";
    data += "s=Inferno Ephone Session\r\n";
    data += "c=IN IP4 "+daddr+"\r\nnt="+string rtime()+" 0\r\nnm=audio "+rtpport+" "+Aproto+" 0\r\nna=rtpmap:0 PCMU/8000\r\nna=pftime:20\r\n";
    data += "c=IN IP4 "+daddr+"\r\nnt=0 0\r\nnm=audio "+rtpport+" "+Aproto+" 0\r\n";
    csp = c.addsession(sid, data);
}
msg := header+string len (array of byte data)+"\r\n\r\n"+data;

if (c.conn == nil) {
    (nil, vaddr, vport, net) := expandnet(proxy(viahost(c, c.tu, 0)));
    if (Dbg) sys->print(Mod+": reconnect to %s at %s!%s!\n", vport, net, vaddr, vport);
    (ok, conn) := dial(net, vaddr, vport, localport(client, vport, vaddr));
    if (ok >= 0) c.conn = ref conn;
}

if (c.conn != nil) {
    if (c.state == "ACK") {
        if (c.addedsessionp()) c.session.startaudio(0);
        c.session.announceaudio();
        c.session.dialaudio();
    }
    if (csp) {
        c.session.startaudio(1);
        c.session.announceaudio();
    }
    if (Vbs) sys->print(Mod+": sending: \r\n%s\r\n", msg);
    fd := c.conn.dfd;
    n := sys->seek(fd, 0, Sys->SEEKSTART);
    if (n < 0) sys->fprint(stderr, Mod+": seek %d %r\n", n);
    n = sys->fprint(fd, "%s", msg);
    if (n < 0) {
        sys->fprint(stderr, Mod+": sending %d %r\n", n);
        c.conn.dfd = nil; c.conn = nil;
        spawn c.resendmsg(client, msg);
    }
    else {
        if (Vbs) sys->print(Mod+": sent: %s\r\n", c.state);
        if (1) c.msg = msg;
        if (restart || c.state == "BYE 200 OK") {
            # This should not be need - possible bug in udp stack...
            spawn restartsip0(client, c);
        }
    }
}
else sys->fprint(stderr, Mod+": send error: mission connection\n");
return c;
}

# derive a taddr for response based on received call data
derive_taddr(c : ref Call) : string
{
    taddr : string;
    (nil, tost) := expand2t(lastel(c.path.via), "\t");
    if (taddr != nil) (taddr, nil) = expand2t(tost, ":");

    if (taddr == nil) {
        tost = sipurlval(c.path.contact);
        if (tost != nil) (nil, taddr, nil) = expand(tost);
    }
    if (taddr == nil)
        (nil, taddr, nil) = expand(c.tu);
    return taddr;
}

# this was needed to work around a vovida problem
add_lport(client : string, flag : int) : string
{
    if (flag) {
        (nil, nil, p, nil) := expandnet(client);
        return explicitport(client, p);
    }
    return client;
}

Call.resend(c : self ref Call, client : string)
{
    if (restartsip(client, c)) {
        if (c.msg != nil)
            c.resendmsg(client, c.msg);
        else if (Dbg) sys->print(Mod+": no message to resend\n");
    }
    c.msg = nil;
}

restartsip0 (client : string, c : ref Call)
{
    restartsip(client, c);
}

Call.resendmsg(c : self ref Call, client : string, msg: string)
{
    if (msg != nil) {
        (nil, vaddr, vport, net) := expandnet(proxy(viahost(c, c.tu, 0)));
        if (c.conn == nil || c.conn.dfd == nil) {
            if (Dbg) sys->print(Mod+": reconnect to %s at %s!%s!\n", vport, net, vaddr, vport);
            (ok, conn) := dial(net, vaddr, vport, localport(client, vport, vaddr));
            if (ok >= 0) {
                c.conn = ref conn;
                fd := c.conn.dfd;
            }
        }
        else {
            sys->fprint(stderr, Mod+": cannot resend\n");
            return;
        }
    }
    fd := c.conn.dfd;
    n := sys->seek(fd, 0, Sys->SEEKSTART);
    if (n < 0) sys->fprint(stderr, Mod+": seek %d %r\n", n);
    n = sys->fprint(fd, "%s", msg);
    if (n < 0) sys->fprint(stderr, Mod+": resending %d %r\n", n);
    else sys->fprint(stderr, Mod+": %s resent\n", c.state);
}
}

```

```

Call.addsession(c : self ref Call, sid, data : string) : int
{
    if (Dbg > 1) sys->print(Mod+": adding session %s\n", data);
    if (c.session == nil)
        c.session = ref Session(sid, data, nil, nil);
    else {
        s := c.session;
        if (s.sid != nil && sid != nil && s.sid != sid) {
            sys->fprintf(stderr, Mod+": changing session id %s->%s\n", s.sid, sid);
            s.sid = sid;
        }
        if (s.data == nil) s.data = data;
        else s.rdata = data :: s.rdata;
        return 1;
    }
    return 0;
}

Call.addedsession(c : self ref Call) : int
{
    s := c.session;
    return (s != nil && s.data != nil && s.rdata != nil);
}

Session.startaudio(s : self ref Session, tipe : int)
{
    data1 := s.data;
    m1 := retrieve("m=", data1);
    c1 := retrieve("c=", data1);
    data2, m2, c2 : string;
    if (Dbg) sys->print(Mod+": session %s data audio:\n%t%s\n", s.sid, m1);
    if (s.rdata != nil) {
        if (Dbg) sys->print("\trdata audio:\n");
        for(l := s.rdata; l != nil; l = tl l) {
            data2 = hd l;
            m2 = retrieve("m=", data2);
            c2 = retrieve("c=", data2);
            if (Dbg) sys->print("\t%t:: %s\n", m2);
            if (m2 != nil) break;
        }
        if (m2 != nil) {
            setupaudio(s, tipe, snth(2, c1), snth(1, m1), snth(2, c2), snth(1, m2), data1, data2);
        }
    }
}

debug := 0;
setupaudio(s : ref Session, tipe : int, faddr, fport, taddr, tport, data1, data2 : string)
{
    if (s.audio != nil) {
        sys->fprintf(stderr, Mod+": audio already started\n");
        return;
    }
    rtcp1 := int fport;
    if (!rtcp1) {
        if (tipe) fport = default_rtpport;
        else fport = default_rrtport;
    }
    rtcp2 := int tport;
    if (!rtcp2) {
        if (tipe) tport = default_rrtport;
        else tport = default_rtpport;
    }
    m1 := retrieve("m=", data1); atype1 := snth(2, m1)+"/"+snth(3, m1);
    m2 := retrieve("m=", data2); atype2 := snth(2, m2)+"/"+snth(3, m2);
    if (start("RTP/AVP", atype1)) rtcp1 = 1 + int fport;
    if (start("RTP/AVP", atype2)) rtcp2 = 1 + int tport;
    if (atype1 != atype2) {
        sys->fprintf(stderr, Mod+": SDP audio negotiation fail: mismatched %s and %s\n", atype1, atype2);
        if (len atype1 > len atype2) atype2 = atype1;
        else atype1 = atype2;
    }
    if (Dbg) sys->print(Mod+": start %s audio: %d %s:%s %s\n", atype1, tipe, faddr, fport, taddr, tport);
    size := 172;
    if (ua != nil) size = ua_seize(size, data1, data2);
    s.audio = ref Audio(faddr+":"+fport+":"+atype1, taddr+":"+tport+":"+atype2, tipe, nil, nil, 0, 0, rtcp1, rtcp2, nil, nil, size, 0);
}

expandatype(t : string) : (string, string, string, string)
{
    (ap, tp, n) := expand3t(t, "/");
    net := "udp";
    if (tp == "TCP") net = "tcp";
    return (net, ap, tp, n);
}

Session.announceaudio(s : self ref Session)
{
    a := s.audio;
    if (a == nil) return;
    if (a.listen) {
        sys->fprintf(stderr, Mod+": audio already announced\n");
        return;
    }
    (faddr, fport, ftype) := expand3t(a.addr1, ":");
    (net1, nil, nil, nil) := expandatype(ftype);
    (taddr, tport, ttype) := expand3t(a.addr2, ":");
    (net2, nil, nil, nil) := expandatype(ttype);
    if (!a.type) {
        if (Laddr != faddr) sys->fprintf(stderr, Mod+": mismatched announce on %s and not %s\n", Laddr, faddr);
        (ok, conn) := announce(net1, "", fport);
        if (ok < 0) {
            sys->fprintf(stderr, Mod+": cannot announce %s\n", fport);
        }
        else {
            if (net1 == "udp") a.conn1 = ref conn;
            ch := chan of int;
            spawn audiolistener(net1, a, conn, ch);
            <- ch;
            if (a.rtcp1) {
                (ok, conn) = announce(net1, "", string a.rtcp1);
                if (ok < 0) {
                    sys->fprintf(stderr, Mod+": cannot announce rtcp %d\n", a.rtcp1);
                }
                else a.cconn1 = ref conn;
            }
        }
    }
    else {
}
}

```

```

if (Laddr != taddr) sys->fprintf(stderr, Mod+": mismatched announce on %s and not %s\n", Laddr, taddr);
(ok, conn) := announce(net2, "", tport);
if (ok < 0) {
    sys->fprintf(stderr, Mod+": cannot announce %s\n", tport);
}
else {
    if (net2 == "udp") a.conn2 = ref conn;
    ch := chan of int;
    spawn audiolistener(net2, a, conn, ch);
    <- ch;
    if (a.rtcp2) {
        (ok, conn) = announce(net2, "", string a.rtcp2);
        if (ok < 0) {
            sys->fprintf(stderr, Mod+": cannot announce rtcp %d\n", a.rtcp2);
        }
        else a.ccon2 = ref conn;
    }
}
}

locaudioport(port : string) : string
{
    if (port == nil) return port;
    return string (int port + 10000);
}

Session.dialaudio(s : self ref Session)
{
    a := s.audio;
    if (a == nil) return;
    if (a.speak) {
        sys->fprintf(stderr, Mod+": audio dial already setup\n");
        return;
    }
    (faddr, fport, ftype) := expand3t(a.addr1, ":");
    (net1, nil, nil, nil) := expanddatatype(ftype);
    (taddr, tport, ttype) := expand3t(a.addr2, ":");
    (net2, nil, nil, nil) := expanddatatype(ttype);
    if (!a.type) {
        ch := chan of int;
        (ok, conn) := dial(net2, taddr, tport, locaudioport(tport));
        if (ok < 0) {
            sys->fprintf(stderr, Mod+": cannot dial %s%s\n", taddr, tport);
        }
    }
    else {
        a.conn2 = ref conn;
        spawn audiospeak(a, conn.dfd, ch);
        <- ch;
        if (a.rtcp2) {
            (ok, conn) = dial(net2, taddr, string a.rtcp2, locaudioport(string a.rtcp2));
            if (ok < 0) {
                sys->fprintf(stderr, Mod+": cannot dial %s%d\n", taddr, a.rtcp2);
            }
            else a.ccon2 = ref conn;
        }
    }
}
else {
    ch := chan of int;
    (ok, conn) := dial(net1, faddr, fport, locaudioport(fport));
    if (ok < 0) {
        sys->fprintf(stderr, Mod+": cannot dial %s%s\n", faddr, fport);
    }
    else {
        a.conn1 = ref conn;
        spawn audiospeak(a, conn.dfd, ch);
        <- ch;
        if (a.rtcp1) {
            (ok, conn) = dial(net1, faddr, string a.rtcp1, locaudioport(string a.rtcp1));
            if (ok < 0) {
                sys->fprintf(stderr, Mod+": cannot dial %s%d\n", taddr, a.rtcp1);
            }
            else a.cconn1 = ref conn;
        }
    }
}
}

audiolistener(net : string, a : ref Audio, c : Sys->Connection, ch : chan of int)
{
    if (net == "udp") {
        audiolisten(a, c.dfd, ch);
        return;
    }
    ch <- a.listen = sys->pctl(0, nil);
    p1 := 0;
    while (a.listen) {
        (ok, nc) := sys->listen(c);
        if (ok < 0) {
            sys->fprintf(stderr, Mod+": listen: %r\n");
            a.listen = 0;
            return;
        }
        buf := array[64] of byte;
        l := sys->open(nc.dir+"remote", sys->OREAD);
        n := sys->read(l, buf, len buf);
        if (n >= 0) {
            if (Dbg) sys->print(Mod+": new audio (%s): %s %s", Mod, nc.dir, string buf[0:n]);
            nc.dfd = sys->open(nc.dir+"/data", sys->ORDWR);
            if (nc.dfd == nil) {
                sys->fprintf(stderr, Mod+": open: %s: %r\n", nc.dir);
                a.listen = 0;
                return;
            }
            if (p1) {
                kill(p1);
                if (Dbg) sys->print(Mod+": kill previous audiolisten %d\n", p1);
            }
            a.conn1 = ref nc;
            nch := chan of int;
            spawn audiolistener(a, nc.dfd, nch);
            p1 = a.listen = <- nch;
            # expect only one attempt for now!
            return;
        }
    }
}

# Sync is used for loop back test of ephone

```

```

# from an emulation version where ua is nil
Sync : chan of array of byte;
audiolisten(a : ref Audio, fd : ref Sys->FD, ch : chan of int)
{
    ok : int; err : string;
    ch <-= a.listen = sys->pctl(0, nil);
    if (!a.size) {
        sys->fprintf(stderr, Mod+ " null buffer size\n");
        return;
    }
    buf := array[a.size] of byte;
    if (Dbg) sys->print(Mod+ " audiolisten start size %d\n", a.size);
    cnt := 0; fnt := 0;
    while(a.listen) {
        n := sys->read(fd, buf, len buf);
        if (n < 0) return;
        else if (n == 0) continue;
        else if (ua != nil) {
            if (a.tipe && !a.busy) {
                if (Dbg) sys->print(Mod+ " audio now busy = %d\n", n);
                a.busy = n;
            }
            (ok, err) = ua->playFrame(buf[0:n]);
            if (ok < 0) break;
            else if (Dbg > 1 && cnt++ > 1000) {
                sys->print(Mod+ " playFrame %dk len %d\n", ++fnt, n);
                cnt = 0;
            }
        }
        # This is the emu to emu sip client test
        else if (n < 30) sys->print(Mod+ " hear: %s\n", string buf[0:n]);
        # This is the ephone to emu loopback test
        else {
            if (Sync == nil) Sync = chan of array of byte;
            Sync <-= buf[0:n];
            if (Dbg && cnt++ > 1000) {
                sys->print(Mod+ " buf len %d\n", n);
                cnt = 0;
            }
        }
    }
    if (Dbg) sys->print(Mod+ " audiolisten end\n");
    if (ok < 0) sys->fprintf(stderr, Mod+ "%s\n", err);
}

audiospeak(a : ref Audio, fd : ref Sys->FD, ch : chan of int)
{
    ch <-= a.speak = sys->pctl(0, nil);
    if (!a.size) {
        sys->fprintf(stderr, Mod+ " null buffer size!\n");
        return;
    }
    buf : array of byte;
    ok := 0; err : string;
    if (ua == nil) {
        (faddr, fport, nil) := expand3t(a.addr1, ":");
        (taddr, tport, nil) := expand3t(a.addr2, ":");
        if (a.tipe) err = faddr+":"+fport;
        else err = taddr+":"+tport;
        buf = array of byte ("test from "+err);
        ok = len buf;
    }
    if (ua != nil) buf = array[a.size] of byte;
    cnt := 0; fnt := 0;
    if (Dbg) sys->print(Mod+ " audiospeak start size %d\n", a.size);
    wait := ua != nil && a.tipe;
    while(a.speak) {
        if (ua != nil) {
            (ok, err) = ua->recordFrame(buf);
            if (ok < 0) break;
            else if (Dbg > 1 && cnt++ > 1000) {
                cnt = 0;
                sys->print(Mod+ " recordFrame %dk len %d\n", ++fnt, ok);
            }
            if (wait && a.busy) wait = 0;
        }
        # This is the emu to ephone loopback test
        else if (Sync != nil) buf = <- Sync;
        # wait for far end to speak first (proxy issue)
        if (wait) continue;
        n := sys->write(fd, buf, ok);
        if (n < 0) return;
        else if (n == 0) continue;
        # This is the emu to emu sip client test
        else if (n < 30) {
            sys->print(Mod+ " speak: %s\n", string buf[0:n]);
            sys->sleep(2000);
        }
    }
    if (Dbg) sys->print(Mod+ " audiospeak end\n");
    if (ok < 0) sys->fprintf(stderr, Mod+ "%s\n", err);
}

Session.endaudio(s : self ref Session)
{
    if (s == nil) return;
    a := s.audio;
    if (a != nil) {
        if (Dbg) sys->print(Mod+ " stop audio: %d %s %s\n", a.tipe, a.addr1, a.addr2);
        pid1 := a.listen;
        pid2 := a.speak;
        a.listen = 0;
        a.speak = 0;
        sys->sleep(200);
        kill(pid1);
        kill(pid2);
        # should not be needed - gc does it
        if (a.conn1 != nil) {a.conn1.dfd = nil; a.conn1 = nil;}
        if (a.conn2 != nil) {a.conn2.dfd = nil; a.conn2 = nil;}
        if (a.cconn1 != nil) {a.cconn1.dfd = nil; a.cconn1 = nil;}
        if (a.cconn2 != nil) {a.cconn2.dfd = nil; a.cconn2 = nil;}
        if (ua != nil) ua_release();
    }
    s.audio = nil;
}

Idkey : con 22e+07;
sid2callid(sid : string) : string
{

```

```

        return string (int sid - int Idkey);
    }

callid2sid(cid : string) : string
{
    return string (int cid | int Idkey);
}

# Preset the audio connections for rtp/tcp tunnelling

Aconn2 : adt
{
    tcpcl1 : Sys->Connection;
    tcpcl2 : Sys->Connection;
};

Audio2 : ref Aconn2;

tcpaudio()
{
    if (Aproto == "RTP/TCP" && Audio2 == nil) {
        (nil, tcpcl1) := announce("tcp", "", Rtpport);
        (nil, tcpcl2) := announce("tcp", "", Rrtport);
        Audio2 = ref Aconn2(tcpcl1, tcpcl2);
    }
}

tcpclear()
{
    Audio2 = nil;
}

announce(net, addr, port : string) : (int, Sys->Connection)
{
    if (net == "tcp" && Audio2 != nil) {
        if (Dbg) sys->print(Mod+": tcp mode with Audio2 present port %s\n", port);
        if (port == Rtpport) return (0, Audio2.tcpcl1);
        else if (port == Rrtport) return (0, Audio2.tcpcl2);
    }
    ur := net+"!"+addr+"!"+port;
    (ok, conn) := sys->announce(ur);

    if (ok < 0) {
        sys->fprintf(stderr, Mod+": cannot announce at %s %r\n", ur);
        return (ok, conn);
    }
}

# open the data file for the connection
if (net == "udp") {
    conn.dfd = sys->open(conn.dir+"/data", sys->ORDWR);
    #conn.dfd = sys->open(conn.dir+"/data", sys->OREAD);

    if (conn.dfd == nil){
        sys->fprintf(stderr, Mod+": cannot open file %s/data: %r\n", conn.dir);
        return (-1, conn);
    }
}
if (Dbg) sys->print(Mod+": announced %s %s port %s\n", ur, conn.dir, port);
return (ok, conn);
}

listen(client : string, conn : Sys->Connection, ch : chan of int)
{
    case Transport {
        "UDP" => {
            cl := ref Client(client, 0, 0, 0);
            ch <- = active = sys->pctl(0, nil);
            cl.listen(ref conn, nil);
        }
        * => listener(client, conn, ch);
    }
}

listener(client : string, c : Sys->Connection, ch : chan of int)
{
    if (Dbg) sys->print(Mod+": start tcp listener\n");
    if (ch != nil) ch <- = active = sys->pctl(0, nil);
    else active = sys->pctl(0, nil);
    cl := ref Client(client, 0, 0, 0);
    while (active) {
        (ok, nc) := sys->listen(c);
        if(ok < 0) {
            sys->fprintf(stderr, Mod+": listen: %r\n");
            active = 0;
            continue;
        }
        buf := array[64] of byte;
        l := sys->open(nc.dir+"/remote", sys->OREAD);
        n := sys->read(l, buf, len buf);
        if(n >= 0)
            if (Dbg) sys->print(Mod+": new request (%s): %s %s", Mod, nc.dir, string buf[0:n]);

        nc.dfd = sys->open(nc.dir+"/data", sys->ORDWR);
        if(nc.dfd == nil)
            sys->fprintf(stderr, Mod+": open: %s: %r\n", nc.dir);
        active = 0;
        return;
    }
    cl.active = 0;
    kill(cl.pid);
    cl = ref Client(client, 0, 0, 0);
    spawn cl.listen(ref nc, nch := chan of int);
    <- nch;
    Clist = cl :: Clist;
}
}

Client : adt
{
    url : string;
    pid : int;
    active : int;
    time : int;
    listen : fn(cl : self ref Client, conn : ref Sys->Connection, ch : chan of int);
};

Clist : list of ref Client;
Client.listen(cl : self ref Client, conn : ref Sys->Connection, ch : chan of int)
{

```

```

client := cl.url;
(line, address, port) := expand(client);
cl.active = cl.pid = sys->pctl(0, nil);
if (ch != nil) {
    ch <= cl.pid;
    cl.time = time();
    if (Dbg) sys->print(Mod+": spawn listen process %d\n", cl.pid);
}
fd := conn.dfd;
buf := array[1024] of byte;
while(active && cl.active) {
    if (cl.time) cl.time = time();
    n := sys->seek(fd, 0, Sys->SEEKSTART);
    if (n < 0) sys->fprintf(stderr, Mod+": seek %d %r\n", n);
    n = sys->read(fd, buf, len buf);
    if (n < 0) {
        sys->fprintf(stderr, Mod+": receiving %d %r\n", n);
        cl.active = 0;
        continue;
    }
    if (n > 0) {
        csp := 0;
        if (Vbs) sys->print(Mod+": receiving:\n");
        (hl, data) := decode(string buf[0:n]);
        c := mkcall(hl, data);
        cp := C.find(c.callid); # was C.this
        if (cp != nil) {
            if (cp.expire == 0 || time() < cp.expire) cp.expire = 0;
            if (cp.state == "INVITE 180 Ringing" && c.state == "INVITE") {
                sys->fprintf(stderr, Mod+": received a duplicate INVITE\n");
                spawn cp.resend(client);
                continue;
            }
            cp.store(c);
            if (c.session != nil)
                csp = cp.addsession(c.session.sid, c.session.data);
            c = cp;
            if (C.this != nil && C.this.callid != c.callid)
                if (Dbg) sys->print(Mod+": switching to received call %s->%s\n", C.this.callid, c.callid);
            C.recv = C;
        }
        else {
            C.recv = c;
            if (C.this != nil)
                if (Dbg) sys->print(Mod+": switching to new received call %s->%s\n", C.this.callid, c.callid);
        }
        C.take(c);
        if (cp != nil && cp.endp()) {
            c = cp;
            c.nextstate(client);
            c.session.endaudio();
            C.rem(cp); c = nil;
        }
        else if (C.recv != nil && C.recv.endp()) {
            c = C.recv;
            c.nextstate(client);
            c.session.endaudio();
            C.rem(C.recv); c = nil;
        }
        else if (c != nil) {
            if (c.conn == nil) {
                (nil, vaddr, vport, net) := expandnet(proxy(viahost(c, c.frum, 1)));
                if (vaddr == Laddr && vport == port) {
                    if (Dbg) sys->print(Mod+": will not connect to self %s:%s\n", vaddr, vport);
                    C.rem(c);
                    continue;
                }
                if (Dbg) sys->print(Mod+": connect back to %s at %s:%s\n", vport, net, vaddr, vport);
                (ok, conn) := dial(net, vaddr, vport, localport(client, vport, vaddr));
                if (ok >= 0) c.conn = ref conn;
                else sys->fprintf(stderr, Mod+": connect failed\n");
            }
            C.take(c); C.recv = c;
            if (c.state == "ACK") {
                if (c.addedsessionp()) c.session.dialaudio();
                else sys->fprintf(stderr, Mod+": audio setup is missing\n");
            }
            else if (csp) {
                if (Dbg) sys->print(Mod+": will start audio next...\n");
                Sch <-= ("", 0);
            }
            c.nextstate(client);
        }
    }
}
cl.pid = 0;
}

cleanList(f : int)
{
    r : list of ref Client;
    for (l := Clist; l != nil; l = tl l) {
        cl := hd l;
        if (f || cl.active == 0) {
            if (cl.pid != 0) kill(cl.pid);
        }
        else r = cl :: r;
    }
    Clist = r;
}

viahost(c : ref Call, default : string, rcv : int) : string
{
    vhost := default;
    transport := Transport;
    if (c.path.via == nil) {
        if (c.path.contact != nil) vhost = c.path.contact;
        if (rcv)
            sys->fprintf(stderr, Mod+": error received empty via field - using %s\n", vhost);
        else if (Dbg) sys->print(Mod+": via () - using %s\n", vhost);
    }
    else {
        proto : string;
        (proto, vhost) = expand2t(hd c.path.via, "\t");
        if (Dbg) sys->print(Mod+": via host %s\n", vhost);
        tp := snth_token(2, proto, "/");
        if (tp == nil) {
            sys->fprintf(stderr, Mod+": unexpected transport protocol %s in via field\n", proto);
        }
    }
}

```

```

        else transport = tp;
    }
    (n, a, p) := expand(vhost);
    r := "@"+a+":"+downcase(transport)+"@"+p;
    if (Dbg) sys->print(Mod+": vahost returns %s\n", r);
    return r;
}

Call.nextstate(c : self ref Call, client : string)
{
    case c.state {
        "INVITE" => {
            c.state += " 180 Ringing";
            Sch <-= ("r", -1);
            c.send(client);
        }
        "INVITE 180 Ringing" => {
            c.state = "INVITE 200 OK";
            c.send(client);
        }
        "INVITE 200 OK" => {
            c.state = "ACK";
            c.send(client);
        }
        "ACK" => {
            c.state = "BYE";
            c.send(client);
        }
        "BYE" => {
            c.state += " 200 OK";
            Sch <-= ("", 0);
            c.send(client);
        }
        "CANCEL" => Sch <-= ("", 0);
        * => {
            (method, code, reason) := c.stateinfo();
            if (code < 300) {
                sys->fprintf(stderr, Mod+": state %s %d %s\n", method, code, reason);
            }
            else {
                if (code >= 400)
                    sys->fprintf(stderr, Mod+": error state %s %d %s\n", method, code, reason);
                else if (code >= 300)
                    sys->fprintf(stderr, Mod+": ignored state %s %d %s\n", method, code, reason);
                C.rem(c);
            }
        }
    }
}

Path : adt
{
    contact : string;
    via : list of string;
    route : list of string;
    record : list of string;
};

mkpath(l : list of string) : ref Path
{
    contact := nonull(findval("Contact:" :: "m:" :: nil, l, 0));
    via := nonull(findall("Via:" :: "v:" :: nil, l, 0));
    if (Dbg > 1)
        if (via != nil) sys->print(Mod+": via (%s . len %d)\n", hd via, len via);
        else sys->print(Mod+": via ()\n");
    (nil, route) := sys->tokenize(nonull(findval("Route:", l, 0)), ",");
    route := nonull(findall("Route:", l, 0));
    if (route != nil) {
        route = sipurls(route);
        if (Dbg > 1)
            sys->print(Mod+": route (%s . len %d)\n", hd route, len route);
    }
    (nil, record) := sys->tokenize(nonull(findval("Record-Route:", l, 0)), ",");
    record := nonull(findall("Record-Route:", l, 0));
    if (record != nil) {
        if (Dbg > 1)
            sys->print(Mod+": record-route (%s . len %d)\n", hd record, len record);
        if (route == nil) {
            recurs := sipurls(record);
            if (contact == nil || findl(sipurlval(contact), recurs)) route = reverse(record);
            else route = reverse(mksipurl(contact) :: record);
        }
    }
    return ref Path(contact, via, route, record);
}

mksipurl(s : string) : string
{
    s = trimspace(s);
    if (s == nil) {
        sys->fprintf(stderr, Mod+": bad () argument to mksipurl\n");
        return nil;
    }
    if (start ("<", s) || start ("sip:", s)) return s;
    else return "<sip:"+s+">";
}

trimspace(s : string) : string
{
    a := 0; b := len s;
    for(i := 0; i < b; i++)
        if (s[i] == ' ' || s[i] == '\t') a++;
        else break;
    for(i = b - 1; i > a; i--)
        if (s[i] == ' ' || s[i] == '\t') b = i;
        else break;
    return s[a:b];
}

mkcall(l : list of string, data : string) : ref Call
{
    (nil, l1) := sys->tokenize(hd l, "\t");
    state, substate : string;
    if (l1 != nil) {
        state = hd l1;
        for(l1 = tl l1; l1 != nil; l1 = tl l1)
            substate += " " + hd l1;
    }
    cseq := nonull(findval("CSeq:", l, 0));
    (nil, l1) = sys->tokenize(cseq, "\t");
}

```

```

if (l1 != nil) {
    if (start("SIP/", state)) {
        if (t1 l1 != nil)
            state = hd t1 l1;
    }
    cseq = 12string(l1);
}
if (Dbg > 2) sys->print(Mod+": cseq=%s\n", cseq);
if (!start(" sip:", substate))
    state += substate;

path := mkpath(l1);

frum := sipurlval(findival("From:" :: "f" :: nil, 1, 0));
tu := sipurlval(findival("To:" :: "t" :: nil, 1, 0));
callid := nonull(findival("Call-ID:" :: "i" :: nil, 1, 0));
if (callid != nil) {
    (nil, l1) = sys->tokenize(callid, " \t@");
    # keep the @ on for lss sip proxy
    (nil, l1) = sys->tokenize(callid, " \t");
    if (l1 != nil) callid = hd l1;
}

sid : string;
if (data != nil) {
    p1 := find("o=", data);
    if (p1 < 0) p1 = 0; else p1 = pos(" \t", data, p1);
    if (p1 < 0) p1 = 0;
    p2 := poso('\n', data, p1); if (p2 < 0) p2 = 0;
    (nil, l1) = sys->tokenize(data[p1:p2], " \t\r\n");
    if (l1 != nil) sid = hd l1;
    if (Dbg) sys->print(Mod+": received sid = %s\n", sid);
}
s : ref Session;
if (sid != nil)
    s = ref Session(sid, data, nil, nil);
return ref Call(nil, path, frum, tu, callid, cseq, state, s, 0, nil, 0);
}

sipurls(l : list of string) : list of string
{
    return reverse(revsipurls(l));
}

revsipurls(l : list of string) : list of string
{
    r : list of string;
    for(; l != nil; l = tl l)
        r = sipurlval(hd l) :: r;
    return r;
}

sipurlval_(s : string) : string
{
    su := "<sip:";
    p1 := find(su, s);
    if (p1 < 0) return nil;
    else p1 += len su;
    p2 := poso('>', s, p1);
    if (p2 < 0) p2 = len s;
    else ++p2;
    rs := s[p1:p2];
    if (rs != nil) {
        rl := len rs;
        if (rs[rl -1] == '>') { rl--; rs = rs[0:rl]; }
        else sys->fprintf(stderr, Mod+": sipurl missing > at end of: %s\n", rs);
        if (Dbg > 1) sys->print(Mod+": sipurl: %s\n", rs);
    }
    return rs;
}

sipurlval(s : string) : string
{
    rs := sipurlval_(s);
    if(rs != nil) return rs;
    su := "sip:";
    p1 := find(su, s);
    if (p1 < 0) return nil;
    else p1 += len su;
    p2 := pos(" \t", s, p1); if (p2 < 0) p2 = len s;
    rs = s[p1:p2];
    if (rs != nil) {
        (nil, l1) := sys->tokenize(rs, ";");
        if (Dbg > 1) sys->print(Mod+": sipurl: %s\n", rs);
        return hd l1;
    }
    return rs;
}

decode(s : string) : (list of string, string)
{
    r : list of string;
    data : string;
    p, pn, n : int = 0;
    if (Vbs) sys->print("[");
    while ((p = poso('\r', s, n)) >= 0 || (pn = poso('\n', s, n)) >= 0) {
        if (pn) p = pn;
        if (p > n) r = s[n:p] :: r;
        sl := nonull(getval("Content-Length:", s[n:p], 0));
        if (sl == nil) sl = nonull(getval("L:", s[n:p], 1));
        nc := '\n';
        if (pn) {
            nc = '\r';
            pn = 0;
        }
        if (len s > p+1 && s[p+1] == nc) p++;
        if (Vbs) sys->print("%s", s[n:p+1]);
        n = p = p +1;
        if (sl != nil) {
            l := int sl;
            data = s[n:n+l];
            if (Vbs) sys->print("%s\r\n\r\n", data);
            break;
        }
    }
    if (Vbs) sys->print("]\r\n");
    return (reverse(r), data);
}

dial(net, addr, rport, port : string) : (int, Sys->Connection)

```

```

{
    if (net != "udp") port = nil;
    (ok, conn) := sys->dial(net+"!"+addr+"!"+rport, port);
    if (ok < 0) {
        sys->fprintf (stderr, Mod+": cannot connect to %s!%s!%s %s\n", net, addr, rport, port);
        return (ok, conn);
    }
    if (Dbg) sys->print(Mod+": new connection to %s!%s!%s %s\n", net, addr, rport, port);
    return(ok, conn);
}

# string and list utils

l2string(l1 : list of string) : string
{
    r : string;
    for(;l1 != nil; l1 = tl l1) {
        r += hd l1; if (tl l1 != nil) r += " ";
    }
    return r;
}

lastel(l : list of string) : string
{
    for (; l != nil; l = tl l)
        if (tl l == nil) return hd l;
    if (l != nil) return hd l;
    return nil;
}

snth(n: int, s : string) : string
{
    (nil, 1) := sys->tokenize(s, " \t\r\n");
    return nth(n, 1);
}

snth_token(n: int, s, t : string) : string
{
    (nil, 1) := sys->tokenize(s, t);
    return nth(n, 1);
}

nth(n: int, l : list of string) : string
{
    for(i := 0; l != nil; l = tl l) {
        if (i == n) return hd l;
        i++;
    }
    return nil;
}

expand2(s : string) : (string, string)
{
    return expand2t(s, "::");
}

expand2t(s, t : string) : (string, string)
{
    (n, l) := sys->tokenize(s, t);
    if (l != nil)
        if (tl l != nil)
            return (hd l, hd tl l);
        else return (hd l, nil);
    return (nil, nil);
}

expand3t(s, t : string) : (string, string, string)
{
    (n, l) := sys->tokenize(s, t);
    if (l != nil)
        if (tl l != nil)
            if (tl tl l != nil)
                return (hd l, hd tl l, hd tl tl l);
            else return (hd l, hd tl l, nil);
        else return (hd l, nil, nil);
    return (nil, nil, nil);
}

retrieve(k, s : string) : string
{
    p := find(k, s);
    if (p >= 0) {
        z := poso('\r', s, p);
        if (z < p) z = poso('\n', s, p);
        if (z < p) z = len s;
        return s[p:z];
    }
    return nil;
}

# blank string are nil
nonull(s : string) : string
{
    if (posnot("\t", s, 0) < 0) return nil;
    return s;
}

noNULL(l : list of string) : list of string
{
    r : list of string;
    for (; l != nil; l = tl l)
        if (nonull(hd l) != nil) r = hd l :: r;
    return reverse(r);
}

pos(e : int, s : string) : int
{
    for(i := 0; i < len s; i++)
        if (e == s[i]) return i;
    return -1;
}

posnot(e : int, s : string) : int
{
    for(i := 0; i < len s; i++)
        if (e != s[i]) return i;
    return -1;
}

```

```

poss(t : string, s : string, o : int) : int
{
    if (o < 0) o = 0;
    for(i := o; i < len s; i++)
        for (j := 0; j < len t; j++)
            if (t[j] == s[i]) return i;
    return -1;
}

possnot(t : string, s : string, o : int) : int
{
    if (o < 0) o = 0;
    for(i := o; i < len s; i++)
        for (j := 0; j < len t; j++)
            if (t[j] != s[i]) return i;
    return -1;
}

find(e, s : string) : int
{
    for(i := 0; i < len s - len e; i++) {
        ok := 1;
        for (j := 0; j < len e; j++)
            if (e[j] != s[i+j]) (ok = 0; break);
        if (ok) return i;
    }
    return -1;
}

findval(k : string, l : list of string, mc : int) : string
{
    r : string;
    for(; l != nil; l = tl l)
        if ((r = getval(k, hd l, mc)) != nil) break;
    return r;
}

findlval(kl : list of string, l : list of string, mc : int) : string
{
    r : string;
    for(; l != nil; l = tl l)
        if ((r = getlval(kl, hd l, mc)) != nil) break;
    return r;
}

findall(k : string, l : list of string, mc : int) : list of string
{
    r : list of string;
    for(; l != nil; l = tl l)
        if ((e := getval(k, hd l, mc)) != nil) r = e :: r;
    return reverse(r);
}

findlall(kl : list of string, l : list of string, mc : int) : list of string
{
    r : list of string;
    for(e := ""; l != nil; l = tl l)
        if ((e = getlval(kl, hd l, mc)) != nil) r = e :: r;
    return reverse(r);
}

findl(e : string, l : list of string) : int
{
    for(; l != nil; l = tl l) if (e == hd l) return 1;
    return 0;
}

reverse(l : list of string) : list of string
{
    r : list of string;
    for(; l != nil; l = tl l) r = hd l :: r;
    return r;
}

poso(c : int, s : string, o : int) : int
{
    for(i := o; i < len s; i++)
        if (s[i] == c) return i;
    return -1;
}

start(k, s : string) : int
{
    if (len s >= len k && k == s[0:len k])
        return 1;
    return 0;
}

# mc = 1 to match case
getval(k, s : string, mc : int) : string
{
    if (len s < len k) return nil;
    if (mc) {
        if (k == s[0:len k]) return s[len k:];
    }
    else {
        if (equalp(k, s[0:len k])) return s[len k:];
    }
    return nil;
}

getlval(kl : list of string, s : string, mc : int) : string
{
    for (; kl != nil; kl = tl kl)
        if ((r := getval(hd kl, s, mc)) != nil) return r;
    return nil;
}

equalp(x, y : string) : int
{
    if (len x != len y) return 0;
    for (i := 0; i < len x; i++)
        if (cupcase(x[i]) != cupcase(y[i])) return 0;
    return 1;
}

cupcase(c : int) : int
{
    if ('a' <= c && c <= 'z') return c + 'A' - 'a';
    else return c;
}

```

```

}

downcase(s : string) : string
{
    for (i := 0; i < len s; i++) {
        c := s[i];
        if ('A' <= c && c <= 'Z') s[i] = c + 'a' - 'A';
    }
    return s;
}

upcase(s : string) : string
{
    for (i := 0; i < len s; i++) {
        c := s[i];
        if ('a' <= c && c <= 'z') s[i] = c + 'A' - 'a';
    }
    return s;
}

# Read list from file

readlist(path : string) : list of string
{
    (ok, dir) := sys->stat(path);
    if (ok < 0) {
        sys->fprintf(stderr, Mod+": stat %s: %r\n", path);
        return nil;
    }
    shfd := sys->open(path, sys->OREAD);
    if (shfd == nil) {
        sys->fprintf(stderr, Mod+": open %s: %r\n", path);
        return nil;
    }
    lc := dir.length;
    if (lc == 0) return nil;

    buf := array[lc] of byte;
    m := 0; n := lc;
    while ((n = sys->read(shfd, buf[m:], lc - m)) > 0)
        m += n;
    if (n < 0) {
        sys->fprintf(stderr, Mod+": read %s: %r\n", path);
        if (!m) return nil;
    }
    if (Dbg > 4) sys->print(Mod+": buf[%d]=%s\n", m, string buf);
    (nil, r) := sys->tokenize(string buf[0:m], "\t\r\n");
    return r;
}

writeln(path : string, l : list of string)
{
    fd := sys->open(path, Sys->OWRITE|Sys->OTRUNC);
    if (fd == nil)
        fd = sys->create(path, Sys->ORDWR, 8r666);
    if (fd == nil) {
        sys->fprintf(stderr, Mod+": %s: %r\n", path);
        return;
    }
    sys->seek(fd, 0, Sys->SEEKSTART);
    for (; l != nil; l = tl l)
        sys->fprintf(fd, "%s\n", hd l);
}

# Append to file

fappend(path : string, more : string)
{
    fd := sys->open(path, Sys->OWRITE);
    if (fd == nil)
        fd = sys->create(path, Sys->ORDWR, 8r666);
    if (fd == nil) {
        sys->fprintf(stderr, Mod+": %s: %r\n", path);
        return;
    }
    sys->seek(fd, 0, Sys->SEEKEND);
    sys->fprintf(fd, "%s\n", more);
}

##### Shannon ephone specific code #####
# Keypad access on shannon ephone

Dupdsp : con "dsp2mp_dup";

listenkeys(ch: chan of int)
{
    ch <- = Epid = sys->pctl(0, nil);

    # Checking for a non existing /dev entry after UCBaudio causes kernel dump!
    fd := sys->open( "/tmp/" + Dupdsp, sys->OREAD );

    # if Watch provides a duplicate channel - use it first
    # else open the DSP device
    if (fd == nil) {
        fd = sys->open( "/dev/dsp2mp", sys->OREAD );
        if (Dbg) sys->print(Mod+": using /dev/dsp2mp\n");
    }
    else
        if (Dbg) sys->print(Mod+": using /tmp/%s from Watch.\n", Dupdsp);

    if(fd == nil) {
        sys->fprintf(stderr, Mod+": cannot open /dev/dsp2mp\n");
        return;
    }

    sfd := sys->open(mp + "/" + sipsrv, Sys->OWRITE);
    if (sfd == nil) {
        sys->fprintf(stderr, Mod+": open %s/%s: %r\n", mp, sipsrv);
        return;
    }
    keywatch(fd, sfd);
}

keywatch(fd, sfd : ref Sys->FD
{
    # See shannon/app1/tel/watch.m

    DSP_KEYPRESS      : con 68;
    # HSET_IN_USE_MSG : con 'o';
}

```

```

    # HSET_NOT_IN_USE_MSG : con 'p';
HSIU : con 'o';
HSNIU : con 'p';
SPKIU : con 's';
SPKNIU : con 't';
hsiu := 0;
spkiu := 0;

buf := array[64] of byte;
n := 0;
while (Epid) {
    n = sys->read(fd, buf, len buf);
    if (n <= 0) continue;
    case int buf[0] {
        HSIU => {
            hsiu = 1;
            if (!spkiu) machine(sfd, "a");
        }
        SPKIU => {
            spkiu = 1;
            if (!hsiu) machine(sfd, "a");
        }
        HSNIU => {
            hsiu = 0;
            if (!spkiu) machine(sfd, "z");
        }
        SPKNIU => {
            spkiu = 0;
            if (!hsiu) machine(sfd, "z");
        }
        DSP_KEYPRESS =>
            if (keydigitp(c := int buf[1])) machine(sfd, sys->sprint("%c", c));
    }
}
if (Dbg) sys->print(Mod+": listenkeys: keywatch end\n");
}

keydigitp(c : int) : int
{
    return (c >= '0' && c <= '9') || c == '#' || c == '*' || c == 'f';
}

# Shannon ephone sound effect FSM

State : adt
{
    s : string;
    d : string;
    c : int;
    f : string;
};

# Default digits collected
Digcnt := 4;

Call.activevp(c : self ref Call) : int
{
    if (c == nil) return 0;
    (t, n, m) := c.stateinfo();
    return t != "REGISTER" && (n >= 0 && n < 300);
}

S : ref State;
machine(fd : ref Sys->FD, c : string)
{
    if (S == nil) S = ref State(nil, nil, 0, nil);
    if (Dbg) sys->print(Mod+": key %s\n", c);
    Sch <-= ("", 0);
    case c {
        "a" => {
            S.s = c;
            if (C.recv.activep() || C.this.activep()) {
                fprints(fd, S.s);
                S.c = 0;
                S.d = nil;
                S.s = "ok";
            }
            else {
                S.c = Digcnt;
                Sch <-= (c, -1);
            }
        }
        "#" => {
            if (S.s == "a") {
                S.c = Digcnt;
                S.d = nil;
                Sch <-= (c, Times);
                sys->sleep(100);
            }
        }
        "z" => {
            S.s = "z";
            S.c = 0;
            S.d = nil;
            Sch <-= ("", 0);
            fprints(fd, S.s);
        }
        "*" => {
            if (S.s == "a") {
                S.d += c;
                S.c--;
                Sch <-= (c, Times);
                sys->sleep(100);
                if (!S.c) {
                    Sch <-= ("", 0);
                    fprints(fd, S.s + S.d);
                    S.s = "invite";
                }
            }
        }
    }
    fprints(fd : ref Sys->FD, s : string)
    {
        b := array of byte s;
        sys->write(fd, b, len b);
    }
}

# Audio conversation support
ua_seize(size : int, data1, data2 : string) : int

```

```

{
    m1 := retrieve("m=", data1); atype1 := snth(2, m1); an1 := snth(3, m1);
    m2 := retrieve("m=", data2); atype2 := snth(2, m2); an2 := snth(3, m2);
    if (start("RTP/", atype1) && start("RTP/", atype2) && an1 == "0" && an2 == "0") {
        # Seize the sound system -- disable all sound effects
        Sch <-= ('>', 0);
        # Only case Rch is used: synchronize with sound muted
        <- Rch;
        rtpmap := snth(1, retrieve("a=rtpmap:", data1));
        (nil, ptme) := expand2t(retrieve("a=ptime:", data1), ":");
        atype := audiotype(rtpmap, ptme);
        ua->setAudioFormat(atype, 1, 8, 12);
        (ok, reason) := audio2open();
        if (ok < 0) sys->fprintf(stderr, Mod+": %s\n", reason);
        else {
            if (Dbg) sys->print(Mod+": UCBAudio open %s %s format buffer size %d\n", rtpmap, ptme, ok);
            size = ok;
            if (debug) {
                checkua();
                looptest(size, 1000);
            }
        }
    }
    else sys->fprintf(stderr, Mod+": cannot negotiate audio %s %s\n", atype1, atype2);
    return size;
}

ua_release()
{
    (ok, reason) := ua->audioClose();
    if (ok < 0) sys->fprintf(stderr, Mod+": %s\n", reason);
    else if (Dbg) sys->print(Mod+": UCBAudio closed\n");
    Sch <-= ('<', 0);
}

audiotype(rtpmap, ptme : string) : int
{
    case rtpmap {
        "PCMU/8000" =>
            case ptme {
                "10" => return UCBAudio->G711MULAWF10;
                "15" => return UCBAudio->G711MULAWF15;
                "20" => return UCBAudio->G711MULAWF20;
                "25" => return UCBAudio->G711MULAWF25;
                "30" => return UCBAudio->G711MULAWF30;
            }
        "PCM/8000" =>
            case ptme {
                "10" => return UCBAudio->PCM8000F10;
                "15" => return UCBAudio->PCM8000F15;
                "20" => return UCBAudio->PCM8000F20;
                "25" => return UCBAudio->PCM8000F25;
                "30" => return UCBAudio->PCM8000F30;
            }
        * => sys->fprintf(stderr, Mod+": cannot set this audio %s %s\n", rtpmap, ptme);
    }
    return UCBAudio->G711MULAWF20;
}

checkua()
{
    # not sure why info is needed in the ua api!
    # info := ua->AudioFormatInfo(0,0,0,0,0,0);
    info := ua->AudioFormatInfo(UCBAudio->G711MULAWF20, 8000, 20, 160, 1, 12, 8);
    (ok, reason) := ua->getAudioParams(ref info);
    sys->print(Mod+": UCBAudio audio params %d %s\n", ok, reason);
    sys->print(Mod+": FormatID(%d)\nSampleRate(%d)\nFrameSize(%d)\nFrameBufSize(%d)\nProtocol(%d)\nFrameHeader(%d)\nJitterBuffer(%d)\n", info..);
    (ok, reason) = ua->getSpeakerVol();
    sys->print(Mod+": UCBAudio speaker volume %d %s\n", ok, reason);
    (ok, reason) = ua->getMicGain();
    sys->print(Mod+": UCBAudio mic gain %d %s\n", ok, reason);
}

looptest(size, max : int)
{
    if (ua == nil) return;
    buf := array[size] of byte;
    ok : int; err : string;
    for(i := 0; i < max; i++) {
        (ok, err) = ua->recordFrame(buf);
        if (ok < 0) break;
        (ok, err) = ua->playFrame(buf);
        if (ok < 0) break;
    }
    if (ok < 0) sys->fprintf(stderr, Mod+": error: %s\n", err);
}

# Serialized sound effect processor
# Sch is the only allowed interface channel
# to the sound system above this layer

Sch : chan of (string, int);
Rch : chan of (string, int);
Spid := 0;
sound(ch : chan of int)
{
    Sch = chan of (string, int);
    Rch = chan of (string, int);
    ch <-= Spid = sys->pctl(0, nil);
    mute := 0;
    while (Spid) {
        (c, n) := <- Sch;
        case c {
            ">" => (stopsound(); mute = 1; Rch <-= (c, n));
            "<" => mute = 0;
            * => if (!mute) {
                if (Dbg) sys->print(Mod+": sound received (%s, %d)\n", c, n);
                startsound(c, n);
            }
        }
    }
    if (Dbg) sys->print(Mod+": sound process ends\n");
}

Soundir : con */sounds/*;
startsound(c : string, n : int)
{
    stopsound();
    f := Soundir;
    case c {

```

```

    " " => return;
    "a" => f += "dialtoneseg.pcm";
    "b" => f += "busy.pcm";
    "c" or "f" => f += "click.pcm";
    "x" => f += "fastbusy.pcm";
    "r" => f = Ringer;
    "w" => f += "ringback.pcm";
    " " =>
        if (Soundp < 2) return;
        else if (len c == 1 && keydigntp(int c[0]))
            f += "dtmf"+c+".pcm";
        else f += c;
    }
    if (Dbg) sys->print(Mod+": f=%s\n", f);
    S.f = f;
    play(f :: string n :: nil);
}

stopsound()
{
    if (S == nil) S = ref State(nil, nil, 0, nil);
    f := S.f;
    S.f = nil;
    if (f != nil) stop(f :: "waitstop" :: nil);
}

killsound()
{
    pid := Spid;
    spawn sendSch("", 0, ch := chan of int);
    killer := <- ch;
    Spid = 0;
    sys->sleep(100);
    kill(pid);
    kill(killer);
}

sendSch(s : string, n : int, ch : chan of int)
{
    if (ch != nil) ch <- = sys->pctl(0, nil);
    Sch <- = (s, n);
}

# Extra ephone and testing testing and debugging

test(args : list of string)
{
    case hd args {
        "d" or "debug" => debug = int hd tl args;
        "p" or "play" or "s" or "stop" => {
            args = tl args;
            ns := "1";
            if (tl args != nil) ns = hd tl args;
            Sch <- = (hd args, int ns);
        }
        "m" or "set" => set(tl args);
        * => usage2();
    }
}

usage2()
{
    sys->print("other options to /tmp/sc:\n\ttd or debug\n\tor play or s or stop\n\t= audio or init or proxy or sound or times or timeout or "
}

Ua : UCBAudio;
set(l : list of string)
{
    if (len l < 2) return;
    case hd l {
        "audio" => Default_audio = tl l;
        "aproto" => {
            Aproto = hd tl l;
            if (Aproto == "nil") {
                Aproto = default_aproto;
                tcpclear();
            }
            else if (Aproto == "RTP/TCP") tcpaudio();
        }
        "proxy" => {
            Proxy = hd tl l;
            if (Proxy == "nil") Proxy = nil;
            else if (Registrar == nil) Registrar = Proxy;
        }
        "regis" => {
            Registrar = hd tl l;
            if (Registrar == "nil") Registrar = nil;
        }
        "sound" => Soundp = int hd tl l;
        "rtpport" => {
            Rtpport = hd tl l;
            if (Rtpport == "nil") Rtpport = default_rtpport;
        }
        "rrtpor" => {
            Rrtport = hd tl l;
            if (Rrtport == "nil") Rrtport = default_rrtport;
        }
        "times" => Times = int hd tl l;
        "timeout" => Timeout = int hd tl l;
        "ua" => {
            if (hd tl l == "nil") {
                if (Ua == nil) Ua = ua; ua = nil;
            }
            else if (Ua == nil) {
                Ua = ua; ua = Ua;
            }
        }
        * => return;
    }
    if (hd l == "audio") sys->print(Mod+": %s = %s", hd l, ls(tl l));
    else sys->print(Mod+": %s = %s\n", hd l, hd tl l);
}

ls(l : list of string) : string
{
    r := "(";
    s := "";
    for (; l != nil; l = tl l) {
        r += hd l;
        if (tl l != nil) r += " ";
    }
}

```

```

        }
        return r + ")";
    }

Default_audio : list of string;
Times := 21;

play(args : list of string)
{
    if (ua == nil) return;
    if (args == nil) return;

    f := hd args;
    times := Times;
    args = tl args;
    if (args != nil) (times = int hd args; args = tl args;)

    if (f == Ringer) {
        ch := chan of int;
        spawn ringing(soundcache(f, nil), times, ch);
        <- ch;
        return;
    }
    (typ, proto, jitter, header) := audioinfo();
    if (typ == 0) {
        (name, ext) := expand2t(f, ".");
        typ = audioformat(ext);
    }
    if (typ != 0) {
        ch := chan of int;
        spawn playsound(soundcache(f, typ :: proto :: jitter :: header :: nil), times, ch);
        <- ch;
    }
    else sys->fprintf(stderr, Mod+"+ cannot play sample of type %d\n", typ);
}

stop(args : list of string)
{
    if (ua == nil) return;
    if (args == nil) return;
    s := soundcache(hd args, nil);
    if (s == nil) sys->fprintf(stderr, Mod+": sound not found %s\n", hd args);
    else if (s.state == 0) sys->fprintf(stderr, Mod+": not playing %s\n", s.name);
    else {
        audiop := s.name != Ringer;
        if (tl args != nil) {
            pid := s.state;
            s.state = 0;
            if (Soundp >= 0)
                timeoutkill(pid, 250, 10, audiop);
        }
        else {
            if (Soundp >= 0)
                spawn timeoutkill(s.state, 1500, 200, audiop);
            s.state = 0;
        }
    }
}

timeoutkill(pid, timeout, quantum, audiop : int)
{
    pstat := "/proc/"+string pid+{/status";
    nc := timeout/quantum;
    while(sys->open(pstat, sys->OREAD) != nil) {
        sys->sleep(quantum);
        if (nc-- <= 0) {
            if (Dbg) sys->print(Mod+": timeout killing %d\n", pid);
            kill(pid);
            if (audiop && ua != nil) ua->audioClose();
            return;
        }
    }
    if (Dbg) sys->print(Mod+": process %d is done\n", pid);
}

Sound : adt
{
    buf : array of byte;
    name : string;
    state : int;
    info : list of int;
};

Cache : list of ref Sound;

soundcache(f : string, info : list of int) : ref Sound
{
    buf : array of string;
    for l := Cache; l != nil; l = tl l)
        if ((hd l).name == f) return (hd l);
    # Artificial reference to Ringer
    if (f == Ringer) {
        Cache = (s := ref Sound(nil, Ringer, 0, info)) :: Cache;
        return s;
    }
    if (info == nil) return nil;
    (ok, dir) := sys->stat(f);
    if (ok < 0) {
        sys->fprintf(stderr, Mod+": stat %s: %r\n", f);
        return nil;
    }
    else {
        fd := sys->open(f, Sys->OREAD);
        n := dir.length;
        buf := array[n] of byte;
        n = sys->read(fd, buf, n);
        if (n < 0) {
            sys->fprintf(stderr, Mod+": read %s: %r\n", f);
            return nil;
        }
        if (n != dir.length) buf = buf[0:n];
        Cache = (s := ref Sound(buf, f, 0, info)) :: Cache;
        return s;
    }
}

audio2open() : (int, string)
{
    (ok, reason) := ua->audioOpen();
    if (ok < 0) {

```

```

sys->fprintf(stderr, Mod+": %s\n", reason);
(ok, reason) = ua->audioClose();
if (ok < 0) sys->fprintf(stderr, Mod+": %s\n", reason);
(ok, reason) = ua->audioOpen();
}
return (ok, reason);
}

Soundp := 1;
playsound(s : ref Sound, times : int, ch : chan of int)
{
    ch <- = sys->pctl(0, nil);
    if (s == nil) {
        sys->fprintf(stderr, Mod+": sound sample not found\n");
        return;
    }
    buf := s.buf;
    n := len buf;
    info := s.info;
    (typ, proto, jitter, header) := values4(info);
    if (Dbg) sys->print(Mod+": open %s - %d %d %d %d\n", s.name, typ, proto, jitter, header);

    if (Soundp < 0) return;
    ua->setAudioFormat(typ, proto, jitter, header);
    (ok, reason) := audio2open();
    if (ok < 0) sys->fprintf(stderr, Mod+": %s\n", reason);
    else {
        fs := ok;
        if (Dbg) sys->print(Mod+": playsound %s size %d nframes %d times %d - %d %d %d %d\n", s.name, fs, n/fs, times, typ, proto, jitter, header);
        s.state = sys->pctl(0, nil);
        while (Soundp > 0 && times--) {
            for(i := 0; i < n - fs; i += fs)
                if (!s.state) break;
            else {
                (ok, reason) = ua->playFrame(buf[i:i+fs]);
                if (ok < 0) break;
            }
            sys->sleep(100);
        }
        if (ok < 0) sys->fprintf(stderr, Mod+": %s\n", reason);
    }
    (ok, reason) = ua->audioClose();
    if (ok < 0) sys->fprintf(stderr, Mod+": %s\n", reason);
}
values4(l : list of int) : (int, int, int, int)
{
    if (len l == 4) return (hd l, hd t1 l, hd t1 t1 l, hd t1 t1 t1 l);
    return (0, 0, 0, 0);
}

audioinfo() : (int, int, int, int)
{
    l := Default_audio;
    if (len l == 3) return (0, int hd l, int hd t1 l, int hd t1 t1 l);
    if (len l > 3) return (audioformat(hd l), int hd t1 l, int hd t1 t1 l, int hd t1 t1 t1 l);
    return (0, 0, 0, 0);
}

audioformat(ext : string) : int
{
    case ext {
        "pcm" => return UCB(Audio->PCM8000F30);
        "pcm20" => return UCB(Audio->PCM8000F20);
        "pcm10" => return UCB(Audio->PCM8000F10);
        "ulw" => return UCB(Audio->G711MULAWF30);
        "ulw20" => return UCB(Audio->G711MULAWF20);
        "ulw10" => return UCB(Audio->G711MULAWF10);
    }
    return UCB(Audio->G711MULAWF20);
}

Ringer : con "ringer";
ringing(s : ref Sound, times : int, ch : chan of int)
{
    if (s == nil) return;
    ch <- = s.state = sys->pctl(0, nil);
    fd := sys->open("/dev/touch2dsp", Sys->OWRITE);
    while(times-- && s.state)
        if (sys->write(fd, array of byte Ringer, len Ringer) <= 0) {
            sys->fprintf(stderr, Mod+": cannot ring this phone\n");
            return;
        }
    else
        for(i := 0; i < 20 && s.state ; i++)
            sys->sleep(200);
}

# Added to restart device remotely
restartdevice(ctxt : ref Draw->Context)
{
    shipcleanup(ctxt);
    fdrb := sys->open("/dev/sysctl", Sys->OWRITE);
    if(fdrb != nil)
        sys->fprintf(fdrb, "reboot");
}

home : con "/usr/inferno/config/*";
tops : con home+"top/scripts/";

ethcon : con "/net/ipifc/0";
route : con "/net/iproute";

shipcleanup(ctxt : ref Draw->Context)
{
    sys->print(Mod+": unconfig ephone connection\n");
    # really should check on /dev/ether0 -- too risky can halt the phone
    (ok, nil) := sys->stat(ethcon);
    if (ok >= 0) {
        (ok, nil) = sys->stat(tops+"eunbind");
        if (ok >= 0)
            shell(ctxt, tops+"eunbind" :: tops+"eunmount" :: tops+"ethenot" :: nil);
        (ok2, nil) := sys->stat(tops+"ethenot");
        if (ok < 0 || ok2 < 0) {
            buf := array[1024] of byte;
            fd := sys->open(ethcon+"status", Sys->OREAD);
            n := sys->read(fd, buf, len buf);
            if (n >= 0) {

```

```

(m, l) := sys->tokenize(string buf[0:n], " \t\n\r");
if (m > 4) {
    eth0 := hd 1;
    if (eth0 != "/dev/ether0") sys->fprint(stderr, Mod+": %s unexpected ether device %s\n", ethcon, hd 1);
    addr := hd tl tl 1;
    mask := hd tl tl tl 1;
    if (mask[0] == '/') mask = "255.255.255.0";
    gway := hd tl tl tl tl 1;
    if (Dbg) sys->print(Mod+": found ether=%s addr=%s mask=%s gway=%s to unconfig\n", eth0, addr, mask, gway);
    fd = sys->open(ethcon+"ctl", Sys->OWRITE);
    if (fd == nil) {
        sys->fprint(stderr, Mod+": %s %r\n", ethcon+"ctl");
        return;
    }
    sys->fprint(fd, "remove %s 255.255.255.0 %s", addr, gway);
    sys->seek(fd, 0, Sys->SEEKSTART);
    sys->fprint(fd, "unbind ether %s", eth0);
    fd = sys->open(route, Sys->OWRITE);
    if (fd == nil) {
        sys->fprint(stderr, Mod+": %s %r\n", route);
        return;
    }
    sys->fprint(fd, "remove 0.0.0.0 0.0.0.0 %s", gway);
    sys->umount("#1", "/dev");
    if (Dbg) sys->print(Mod+": ethernet device unmounted\n");
}
else sys->print(Mod+": ethernet device not active\n");
else sys->fprint(stderr, Mod+": %s %r\n", ethcon+"status");
}
}

shell(ctxt : ref Draw->Context, args : list of string)
{
    sh := load Command Command->PATH;
    if (sh != nil)
        sh->init(ctxt, Command->PATH :: "-n" :: args);
    else
        sys->fprint(stderr, Mod+": %s %r\n", Command->PATH);
}
}

```